Visualizing Fluid Flows Using Line Integral Convolution Method

Shuyang Wang
Mentors: Shravan Veerapaneni, Ruowen Liu

University of Michigan
July 2019

Abstract

We compare the results of visualizing complex fluid flows using the Line Integral Convolution (LIC) method, streamlines and arrows to demonstrate the advantages of the LIC method. We also apply the LIC method to visualize fluids inside an irregular pipe. To deal with the inaccurate velocity data near the fluid-structure interface, we design several methods to interpolate the data between the boundary and the interior part. Barycentric interpolation performs well and is generalizable. We conclude with the visualization of complex fluid flows in 3D.

1 Introduction

Complex fluids are made of simple fluids and particles of other substances contained in them. The interaction between the fluids and the particles produces various interesting features of the fluid flows. Given the data of the vector field and the shapes of the particles, we want to visualize the complex fluids. Common methods of visualizing vector fields, such as using streamlines and using arrows, have some drawbacks. The Line Integral Convolution (LIC) method proposed by Cabral and Leedom[1], however, produces a comprehensible and accurate representation of the vector field. In section 2, we provide an interpretation of the LIC method from the perspective of convolution in image processing. In section 3, we present the result of applying the LIC method to the visualization of an example complex fluid and compare it with the results of using streamlines and of using arrows to demonstrate the advantages of the LIC method. The merits of the LIC method can also be shown in the visualization of fluids inside irregular structures, where the interaction between the fluid and the structure gives rise to complicated movements of the flows. In section 4, we demonstrate the visualization of an example Newtonian fluid inside an irregular pipe and discuss several ways of interpolating the data near the fluid-structure interface. We proceed to the visualization of 3D complex fluid flows in section 5 as a closure to this report.

2 Line Integral Convolution method

The LIC method takes in a white noisy background texture. For each pixel, it integrates the convolution kernel along the streamline and computes the convolution of the input texture and the kernel to get the output value of the pixel. Each pixel is processed in the same way to generate the output texture.

2.1 Convolution in image processing

The idea of convolution in image processing is used in this method. *Convolution* is derived from the mathematical definition of convolution of two functions in discrete case. One function is the representation of the original image, the other function is called a *kernel*. In 1D convolution, let X be the input image, which maps each pixel to its value in the original image, and let K be the kernel. The output value for a pixel m is given by $Y(m) = \sum_i X(i) \cdot K(m-i)$.

2D convolution is more common in image processing. A matrix M of the same size of the original image is convoluted with a kernel K. Each entry $M(i,j)$ is the value of the (i,j)-th pixel of the input image. The output value for a pixel (m,n) is given by $Y(m,n) = \sum_i \sum_j M(i,j) \cdot K(m-i, n-j)$.

Depending on different purposes, a kernel can be chosen to be of different shapes and values to achieve certain effects, such as blurring or sharpening the local features of the input image. In both 1D and 2D cases, the domain of the kernel is usually chosen to be symmetric about the origin. The summation is taken over all the pixels within the domain of the kernel, so that the convolution at pixel p takes a weighted sum of the pixels in a neighborhood centered at p of the size of the kernel domain, where the weights are determined by the kernel. While this method can cause problem for pixels on the edge, where some pixels in its neighborhood are out of bounds of the original image, this issue can be handled by, for example, padding the input image.

2.2 Algorithm of the LIC method

In Cabral and Leedom's method, a matrix X of the same size as the given vector field and filled with random data is used as the original image in convolution. For each pixel, the kernel K is computed from the given vector field V. We call the output matrix Y. The algorithm processes one pixel $p(x, y)$ at a time in the following steps. Note that (x, y) is the Cartesian coordinate of p in the vector field by convention, which is different from the i, j indexing in matrices.

Step 1: Determine the streamline going through $p(x, y)$ of length $2L$.
We call the starting pixel P_0. The vector at this pixel is given by $V(P_0)$. The next pixel in the forward direction of the streamline is the nearest pixel in the direction of $V(P_0)$, and we call it P_1. The length Δs_0 of the streamline segment inside the cell of P_0 is stored. Once the streamline reaches the cell of P_1, it starts to follow the direction of the vector $V(P_1)$, which leads to the next pixel P_2. Repeat the above method until the total length of streamline segments reaches L. To maintain symmetry, we compute the streamline in the backward direction in a similar way. Then we have the streamline $S = \{P'_{k'}, P'_{k'-1}, ..., P'_1, P_0, P_1, ..., P_{k-1}, P_k\}$.

Remark. *As illustrated in Figure 1 from Cabral and Leedom's paper [1], the streamline going through p(x,y) locally follows the direction of the vectors, which is a good approximation of the flow.*
This streamline specifies the domain of the kernel, as the shape of the neighborhood of $p(x, y)$ in convolution. Notice that the streamline centered at p can be different for different pixels, so a specific kernel is computed for each p.

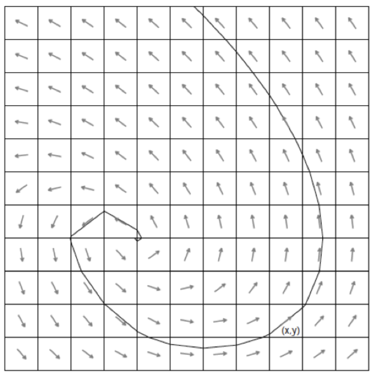


Figure 1: Streamline going through $p(x, y)$[1]

Step 2: Integrate along the streamline.
Once we determine the domain of the kernel, we want to assign weights to the pixels covered in the kernel. A scalar function k that maps a point (x, y) to a real number is integrated along the streamline. Each segment of the streamline can be viewed as a parametric curve in t. Compute $w_i = \int_{s_i} k(t)\, dt$, where s_i is the segment in the cell of pixel P_i. Similarly, w'_i is computed for each segment in the backward direction. Now we define the kernel K to be a function that maps each streamline segment to the weight w.

Remark. *The scalar function k can be customized for different purposes. To visualize the vector field, k can simply be a constant function. In this case, the integral coincides with the length Δs_i (multiplied by the constant) stored before. For the purpose of post processing images with LIC method, special k functions can be used.*

Step 3: Compute the output value for each pixel.
Although the input texture is 2 dimensional, the idea of 1D convolution is used, since the kernel on the streamline can be viewed as a 1 dimensional line consisting of continuous line segments. For the pixel p, $Y(p) = (\sum_i X(P_i) \cdot w_i + \sum_j X(P'_j) \cdot w'_j)/(\sum_i w_i + \sum_j w'_j)$. The numerator is the weighted sum of the pixels on the streamline and the denominator is to normalize the value. Process every pixel in the same way to generate the output texture.

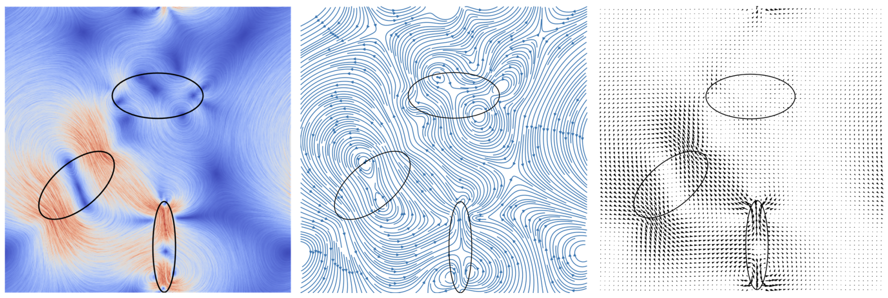
Remark. *The LIC method has the advantage of locality inherited from the property of convolution, which allows the algorithm to be optimized. The accuracy of this method can be illustrated by the good approximation of the streamline as discussed in Step 1.*

2.3 Implementation

The development of our LIC program is based on the implementation of the LIC method in Python by Dzhelil Rufat[2]. We added features such as assigning a color map to indicate the magnitude of the vectors, plotting shapes of the particles and options to smooth the boundary of the pipe and to specify the size of the output image.

3 Visualization of complex fluid flows

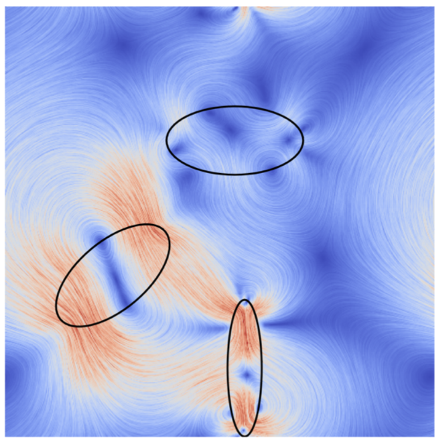
We apply the LIC method to visualize a complex fluid containing three vesicles in continuous time steps. We also use streamlines and arrows to represent the vector field at time step 1. The comparison of the results of the three methods is shown in the figure below.

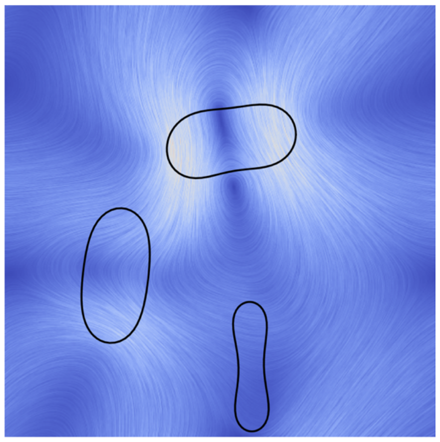


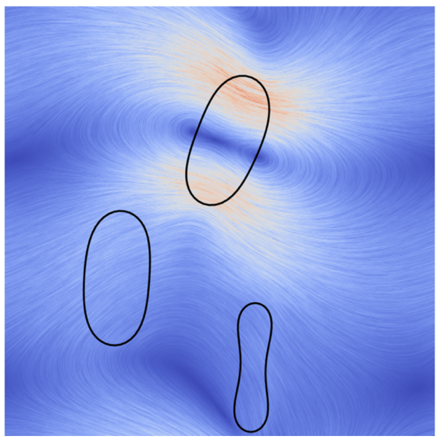
(a) LIC method (b) Streamlines (c) Arrows

Figure 2

The effect of sampling the vector field with streamlines depends on the placement and the density of the samples. It can miss certain features of the flow when no streamline crosses that region. Using arrows at each data point to represent the vectors does not miss data, but can take up a considerable amount of memory to store the information for each arrow, which is highly inefficient when the input size is large. Moreover, the sparse arrows can not clearly illustrate the subtle features of the flow. As a texture-based technique, the LIC method outputs a more comprehensible and accurate visualization of the complex fluids compared to these methods. We normalize the output values of the pixels globally for all time steps and make an animation of the movements of the fluid. The images at time step 1, 51, 101 and 151 are shown below.


(a) Time step 1


(b) Time step 51


(c) Time step 101

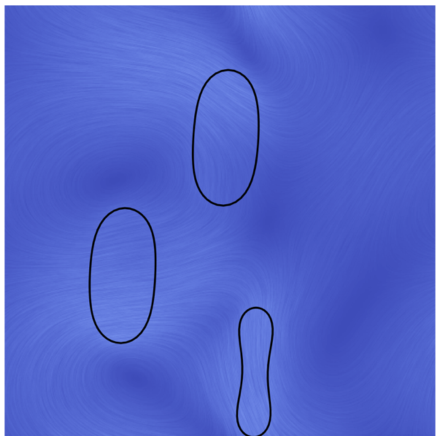

(d) Time step 151

Figure 3

4 Visualization of fluids inside an irregular pipe

The interaction of the fluids and the irregular structure containing it can produce interesting features of the flow. The LIC method can be used to properly visualize such flows. We study a deformable irregular pipe as shown in Figure 4. The data given include the meshgrid of the vector field, the velocity at each grid point, and the coordinates and velocities of the points on the boundary of the pipe.

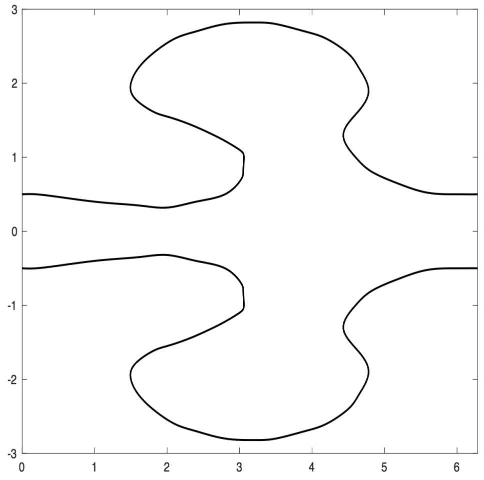


Figure 4: Irregular pipe

The result of applying the LIC method and removing the area outside the pipe is shown below in Figure 5. To deal with the problem of inaccurate data near the fluid-structure interface, we want to interpolate the data in the gap between the boundary and the interior part of the pipe, shown as the green area in Figure 6.

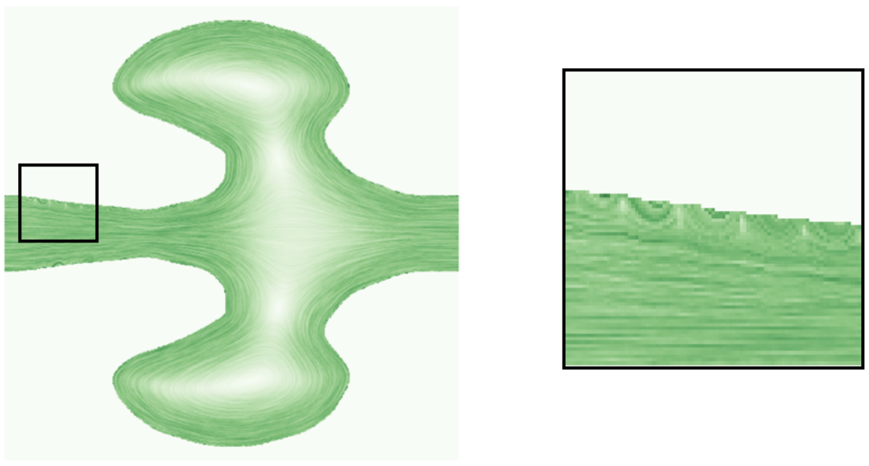


Figure 5

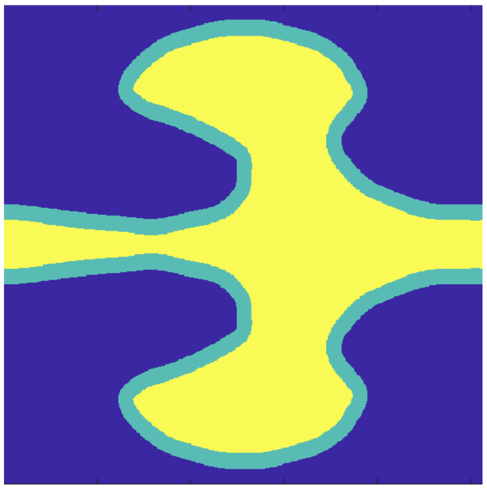


Figure 6

4.1 1D linear interpolation

On each vertical line $x = x_0$, we linearly interpolate the data of $P(x_0, y)$ in the gap using the the nearest interior point $P_1(x_0, y_1)$ and outside point $P_2(x_0, y_2)$ on the vertical line. Since the data of outside points are inaccurate, we use the velocity of the nearest point on the boundary to P_2 as the velocity at P_2. The interpolated values are given by

$$V_x(P) = V_x(P_1)\frac{y_2 - y}{y_2 - y_1} + V_x(P_2)\frac{y - y_1}{y_2 - y_1}$$

$$V_y(P) = V_y(P_1)\frac{y_2 - y}{y_2 - y_1} + V_y(P_2)\frac{y - y_1}{y_2 - y_1}.$$

The result of linear interpolation on the vertical line is shown in Figure 7. This method performs well where the boundary is almost perpendicular to the vertical line, as shown in the black frame. When the point is distant from the nearest points outside the gap on the vertical line, this estimation is unreliable, as shown in the blue frame.

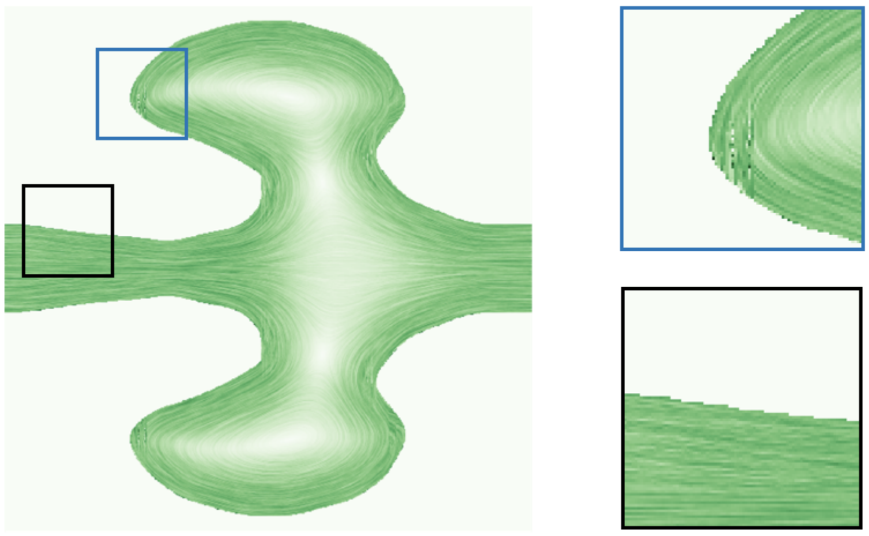


Figure 7: 1D linear interpolation

4.2 2D spline interpolation

A 2D spline is a piecewise polynomial parametric curve consisting of the coordinate functions $x(t)$ and $y(t)$. Given the coordinates and function values of a certain set of points, we can find a spline curve going through them, which can be used to compute the function values for other points. To interpolate the points in the gap, we want to use a set of interior and boundary points to find the spline curve. We first compute the norms at the boundary points. Each time, we use the norms of three consecutive points to define a local region for interpolation. The intersections of the norms with the pipe boundary, the gap boundary, and ten interior lines parallel to the boundary are used to compute the spline function, as shown in Figure 8. The velocities of the gap points in the local region can be obtained from the spline function.

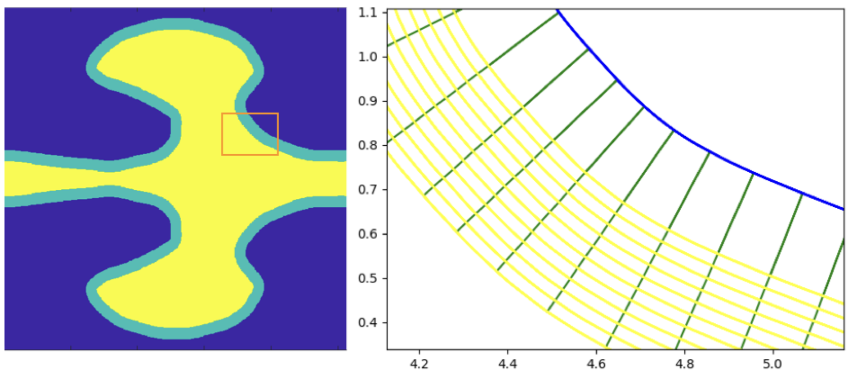


Figure 8: 2D spline interpolation

The result of 2D spline interpolation is shown below in Figure 9. The method performs well in general. It fixes the obvious inaccuracy in the black frame, but the result is not satisfying in certain regions, as in the blue frame. Moreover, this algorithm is hard to implement and needs the derivatives at each boundary points to compute the norms, so it is not ideal to be generalized to other irregular pipes.

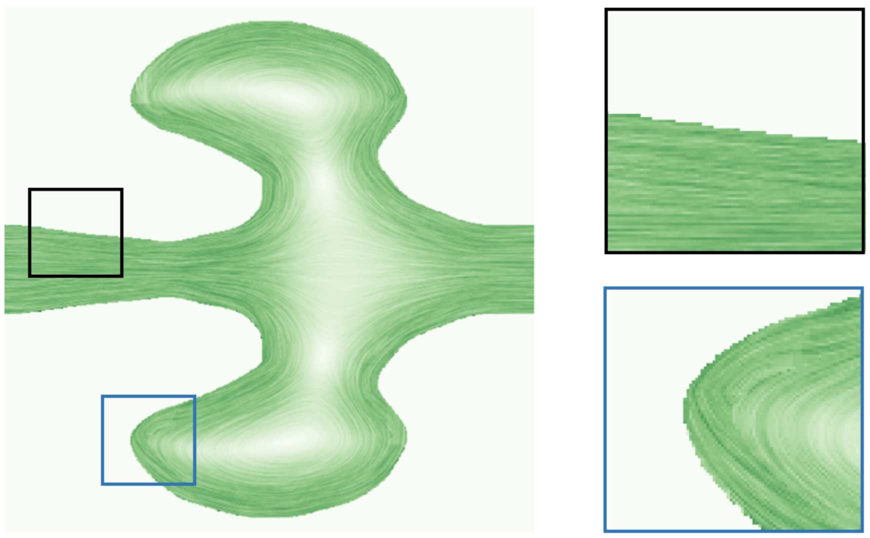


Figure 9: 2D spline interpolation

4.3 Barycentric interpolation

Given a triangle $\triangle A_1A_2A_3$ as in Figure 10, for an arbitrary point P, its Barycentric coordinate is a triple (t_1, t_2, t_3) such that when assigning masses of t_1, t_2, t_3 to A_1, A_2, A_3 respectively, P is the geometric centroid of the triangle. Note that these masses are allowed to take negative values, when P is outside the triangle. We specify an orientation of the triangle, for example, $A_1 \to A_2 \to A_3 \to A_1$, then t_i can be obtained by computing the volume of $\triangle A_jA_kP$, where $j \neq i, k \neq i$ and $A_j \to A_k$ is in the orientation. Suppose the points have coordinates $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, $A_3(x_3, y_3)$ and $P(x, y)$, then

$$t_i = \frac{1}{2}(\overrightarrow{A_jA_k} \times \overrightarrow{A_jP}) = \frac{1}{2}((x_k - x_j)(y - y_j) - (x - x_j)(y_k - y_j)).$$

After normalizing t_1, t_2, t_3, we can compute the value at P by

$$V = t_1 \cdot V_1 + t_2 \cdot V_2 + t_3 \cdot V_3,$$

where V_1, V_2, V_3 are the values at A_1, A_2 and A_3.

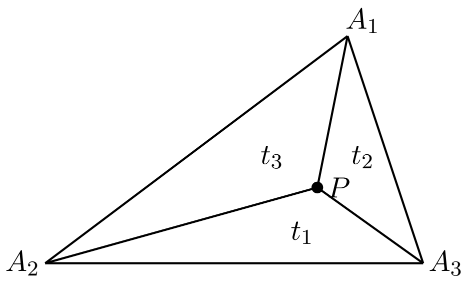


Figure 10

For each point in the gap, we find two nearest points on the boundary and the nearest interior point to form a triangle. We use Barycentric interpolation to compute the velocity at the point in the gap. Barycentric interpolation works well and can be easily generalized to other pipe shapes. We conclude the visualization of 2D fluid flows with the nice result of this method in Figure 11.

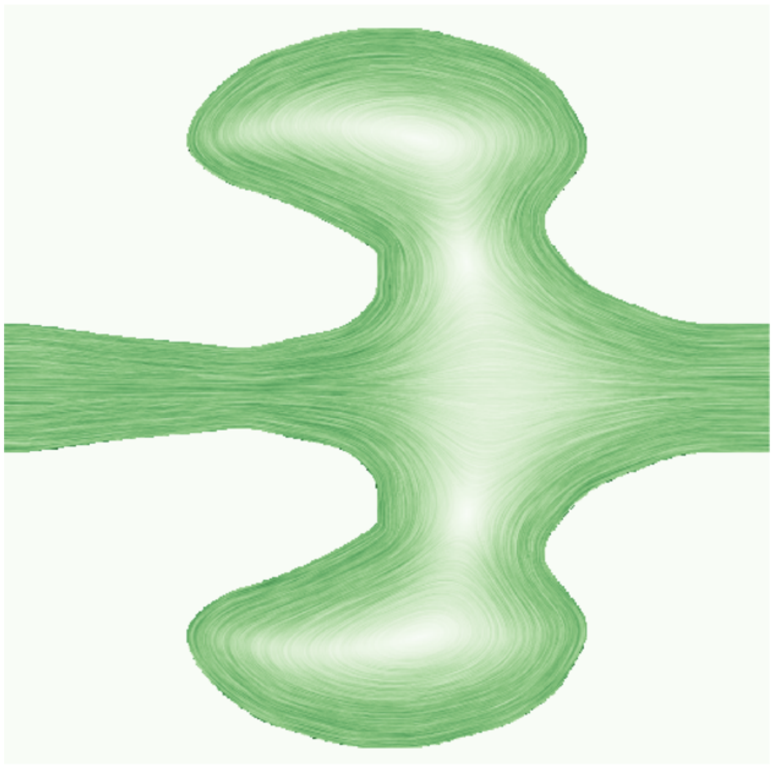
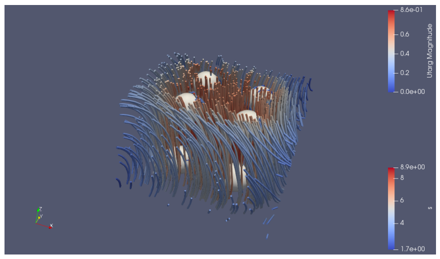


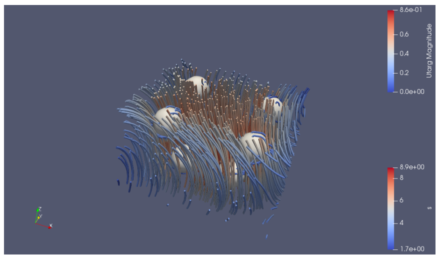
Figure 11: Barycentric interpolation

5 Visualization of complex fluid flows in 3D

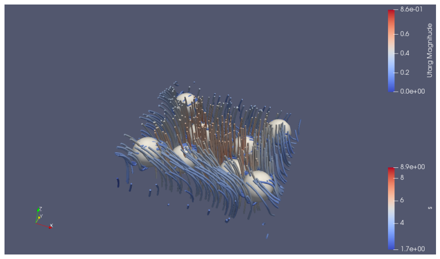
We want to visualize the sedimentation of a cube within the fluid. Although the LIC method is capable of being extended to 3D volume visualization[3], the inherent difficulty lies in visualizing dense 3D textures. We use ParaView instead to visualize this sedimentation process by sampling the vector field with streamlines at each time step. The streamlines are rendered as tubes whose color indicates the magnitude of the velocity and spheres are placed at the eight corners of the cube. Figure 12 presents the images at time 0.0, 50.0, 100.0 and 150.0.



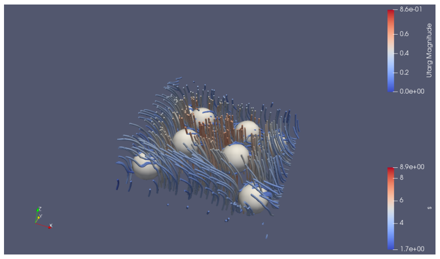
(a) Image at time 0.0



(b) Image at time 50.0



(c) Image at time 100.0



(d) Image at time 150.0

Figure 12: Sedimentation

Future directions To animate the process of the sedimentation, we use a sequence of images at each time step to make a movie. Jobard and Lefer [4] propose a way to correlate the streamlines in neighboring frames to produce a smooth animation of unsteady flow in 2D. This method can be generalized to 3D flows. It requires to keep track of streamlines in each frame, which can be further explored in future work.

References

[1] Cabral, Brian, and Leith Casey Leedom. *Imaging vector fields using line integral convolution.* No. UCRL-JC-112935; CONF-9208240-1. Lawrence Livermore National Lab., CA (United States), 1993.

[2] Rufat, Dzhelil. "LicPy." *LicPy - Dzhelil Rufat*, rufat.be/licpy/

[3] Interrante, Victoria, and Chester Grosch. "Visualizing 3D flow." *IEEE Computer Graphics and Applications* 18.4 (1998): 49-53.

[4] Jobard, Bruno, and Wilfrid Lefer. "Unsteady flow visualization by animating evenly-spaced streamlines." *Computer Graphics Forum.* Vol. 19. No. 3. Oxford, UK and Boston, USA: Blackwell Publishers Ltd, 2000.